ELSEVIER Linear Algebra and its Applications 275–276 (1998) 451–470

LINEAR ALGEBRA AND ITS APPLICATIONS

Parallel codes for computing the numerical rank [1]

Gregorio Quintana-Ortí [*,2], Enrique S. Quintana-Ortí [3]

Universidad Jaime I, Departamento de Informática, Campus Penyeta Roja, E-12071 Castellón, Spain

Received 6 May 1997; accepted 12 August 1997

Submitted by G. Michler

Abstract

In this paper we present an experimental comparison of several numerical tools for computing the numerical rank of dense matrices. The study includes the well-known SVD, the URV decomposition, and several rank-revealing QR factorizations: the QR factorization with column pivoting, and two QR factorizations with restricted column pivoting. Two different parallel programming methodologies are analyzed in our paper. First, we present block-partitioned algorithms for the URV decomposition and rank-revealing QR factorizations which provide efficient implementations on shared memory environments. Furthermore, we also present parallel distributed algorithms, based on the message-passing paradigm, for computing rank-revealing QR factorizations on multicomputers.

1. Introduction

Consider the matrix $A \in \mathbb{R}^{m \times n}$ (w.l.o.g. $m \geq n$), and define its rank as

$$r := \text{rank}(A) = \dim(\text{range}(A)).$$

The rank of a matrix is related to its singular values $\sigma(A) = \{\sigma_1, \sigma_2, \ldots, \sigma_p\}$, $p = \min(m, n)$, since

* Corresponding author. E-mail: gquintan@inj.uji.es.

[1] All authors were partially supported by the Spanish CICYT project TIC 96-1062-C03-03.

[2] This work was partially carried out while the author was visiting the Mathematics and Computer Science Division at Argonne National Laboratory.

[3] E-mail: quintana@inf.uji.es.

0024-3795/98/$19.00
PII: S0024-3795(97)10032-5

$$\operatorname{rank}(A) = r \quad \text{iff} \quad \sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0.$$

In practice, due to rounding and measuring errors, this definition is changed to that of numerical rank [13]. Thus, matrix A has numerical rank (δ, ϵ, r) with respect to $\|\cdot\|$ iff

$$r = \inf\{\operatorname{rank}(B): \|A - B\| \leqslant \epsilon\}$$

and

$$\epsilon < \delta \leqslant \sup\{\eta: \|A - B\| \leqslant \eta \Rightarrow \operatorname{rank}(B) \geqslant r\}.$$

In particular, if the two-norm is used, then A has numerical rank (δ, ϵ, r) iff

$$\sigma_r \geqslant \delta > \epsilon \geqslant \sigma_{r+1}.$$

The computation of the numerical rank of dense matrices is a problem that arises in many areas of science and engineering, for example, in the solution of rank-deficient leastsquares problems, in geodesy [14], computer-aided design [17], nonlinear leastsquares problems [21], the solution of integral equations [10], and the calculation of splines [16]. Other applications arise in beam forming [3], spectral estimation [20], and regularization [18].

Basically, there exist three numerical tools that can be applied to this problem: the singular value decomposition (SVD), the URV decomposition, and the rank-revealing QR (RRQR) factorizations.

The SVD offers highly reliable and complete information about the singular value distribution of a matrix [15]; e.g., the singular values and singular vectors can be explicitly computed, and from these, explicit basis for the range space and the null space of the matrix are available. On the other hand, if the only goal is to compute the numerical rank, most of this information is unnecessary. The SVD presents the major drawback of its high computational cost. Thus, from a practical point of view, this decomposition is only interesting when the accuracy of the results is a major requirement.

An alternative method that allows the identification of the null space of a matrix, and provides an explicit basis for it, is the URV decomposition [26]. This decomposition is as effective as the SVD for separating the null space of a matrix, and allows a low-cost update of the decomposition if one or several rows are added to A once its factorization is known.

A third approach for the computation of the numerical rank are the RRQR factorizations. The most well-known among these is the traditional QR factorization with column pivoting (hereafter, QRP) [11]. Two new RRQR algorithms have been recently developed that improve both the rank-revealing properties and the efficiency of the QRP [4,5].

On current cache-based (parallel) architectures, a key factor to an efficient use of the computer resources is the amount of data movements between the cache memory and the main memory. On these architectures the performance

of the algorithms is increased by reducing the transfers by means of the so-called block-partitioned techniques, and matrix–matrix (BLAS-3) computational kernels.

On parallel distributed architectures (multicomputers) an additional key factor is the amount of data transfers of local data among the processors. Thus, on multicomputers the goal is to combine a small communication overhead, and a balanced distribution of the computational load among the processors. These goals can also be achieved by means of matrix–matrix (BLAS-3) kernels.

In this paper we report the experimental comparison of several algorithms for computing the numerical rank of dense matrices on cache-based architectures (basically, uniprocessors and shared-memory multiprocessors). The study includes the LAPACK implementation of the SVD, a new block-partitioned algorithm for the URV decomposition, a BLAS-3 algorithm for the QRP, and two recent BLAS-3 RRQR factorizations.

We also present an experimental analysis of several parallel distributed algorithms for computing rank-revealing QR factorizations on multicomputers.

In Section 2 the Golub–Reinsch SVD algorithm is described. In Section 3 we introduce a new block-partitioned algorithm for computing the URV decomposition. Then, in Section 4 we present a BLAS-3 version of the traditional QRP factorization, and two recently developed RRQR factorizations. Finally, we compare the performance and accuracy of these numerical tools in Section 5, and in Section 6 we present the concluding remarks.

2. The SVD in LAPACK

In the SVD, matrix $A \in \mathbb{R}^{m \times n}$ is factorized as

$$A = U \Sigma V^{\mathrm{T}}, \tag{1}$$

where $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p) \in \mathbb{R}^{n \times n}$ and $U \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$, have orthonormal columns.

The Golub–Reinsch algorithm for the SVD consists of a two-stage procedure. First, matrix A is bidiagonalized by means of Householder reflectors, as follows.

Algorithm SVD-Stage 1
for $j = 1{:}n$
  Compute a Householder reflector $U_j \in \mathbb{R}^{m-j+1 \times m-j+1}$ such that
    $U_j A(j{:}m, j) = (\|A(j{:}m, j)\|_2, 0, \ldots, 0)^{\mathrm{T}}$
  Apply the Householder reflector to the matrix:
    $A(j{:}m, j{:}n) \leftarrow U_j A(j{:}m, j{:}n)$
  if $j \leqslant n - 2$
    Compute a Householder reflector $V_j \in \mathbb{R}^{n-j \times n-j}$ such that

$$V_j A(j, j+1:n)^{\mathrm{T}} = (\|A(j, j+1:n)\|_2, 0, \ldots, 0)^{\mathrm{T}}$$

Apply the Householder reflector to the matrix:

$$A(j:m, j:n) \leftarrow A(j:m, j+1:n) V_j$$

end if

end for

Then, an iterative procedure is applied to the bidiagonal matrix to annihilate the off-diagonal elements [12]. This procedure relies on the application of Givens rotations from the left and from the right to the bidiagonal matrix, and the problem is usually decoupled into smaller-size problems after a few iterations.

The Golub–Reinsch SVD algorithm provides highly reliable information about the singular value distribution. Actually, the exact SVD of $\tilde{A} = A + \Delta A$, where $\|\Delta A\|_2 \leqslant \epsilon \|A\|_2$ is computed [15]. Furthermore, the computed singular values $\tilde{\sigma}_k$, $k = 1, \ldots, p$, satisfy that $\|\sigma_k - \tilde{\sigma}_k\| < \epsilon\sigma_1$; thus, the numerical rank is computed in the SVD decomposition with a high accuracy.

On the other hand, the computation of the singular values (matrix Σ) by means of the Golub–Reinsch SVD algorithm is quite expensive. It requires $O(4mn^2 - 4n^3/3)$ flops (floating-point arithmetic operations) for the bidiagonalization procedure, and approximately $O(32i)$ flops for each iteration of the iterative refinement on a block of size i [12].

The implementation of the SVD in LAPACK [1] (`xGESVD`) reduces a matrix A to bidiagonal form (`xGEBRD`) by means of block reflectors. At each iteration of the bidiagonalizing procedure, blocks of nb columns and rows of the matrix are bidiagonalized, and then the rest of the matrix is updated with highly efficient BLAS-3 kernels.

In the second stage, an iterative algorithm [8] is used to compute accurately the singular values and is usually faster than the standard approach in the SVD Golub–Reinsch algorithm. In this algorithm (`xDBSQR`), Givens rotations are applied to two vectors, which correspond to the diagonal and nonzero off-diagonal elements of the bidiagonal matrix.

3. The URV factorization

In the URV decomposition [26], $A \in \mathbb{R}^{m \times n}$ is factorized into

$$A = U \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} V^{\mathrm{T}}, \tag{2}$$

where $U \in \mathbb{R}^{m \times n}$ and $V \in \mathbb{R}^{n \times n}$ have orthonormal columns, $R_{11} \in \mathbb{R}^{r \times r}$, and $\|R_{12}\|_2 \approx \|R_{22}\|_2 \approx \sigma_{r+1}$.

The algorithm for the URV decomposition [26] requires an initial triangularization of the matrix, which can be carried out by means of any orthogonal decomposition, e.g., the QR factorization.

Next, the triangular matrix is further reduced to the required form Eq. (2) by applying Givens rotations, as follows.

Algorithm URV-Stage 2
$i \leftarrow n$
while $(\sigma_{\min}(R(1:i, 1:i)) < \text{tol})$
  Compute the right singular vector, $v_i \in \mathbb{R}^i$,
    associated with $\sigma_{\min}(R(1:i, 1:i))$
  for $k = i-1:-1:1$
    Compute a Givens rotation $G^r_{i,k} \in \mathbb{R}^{2\times 2}$ such that
      $G^r_{i,k} v_i(k:k+1) = (\|v_i(k:k+1)\|_2, 0)^T$
    Apply the Givens rotation to the matrix:
      $R(1:k+1, k:k+1) \leftarrow R(1:k+1, k:k+1) G^r_{i,k}$
    Compute a Givens rotation $G^l_{i,k} \in \mathbb{R}^{2\times 2}$ such that
      $G^l_{i,k} R(k:k+1, k) = (\|R(k:k+1, k)\|_2, 0)^T$
    Apply the Givens rotation to the matrix:
      $R(k:k+1, k:n) \leftarrow G^l_{i,k} R(k:k+1, k:n)$
  end for
  $i \leftarrow i-1$
end while

The triangular matrix can be further processed to provide a matrix with dominant diagonal elements [26].

Since only orthogonal transformations are involved in the URV decomposition, we obtain the exact triangular factor R of $\hat{A} = A + \Delta A$, where $\|\Delta A\|_2 \leqslant c\epsilon\|A\|_2$, and c is a constant that depends on m, n, and the number of orthogonal transformations applied [15]. Furthermore, the second stage in the URV decomposition is guaranteed to reveal whether A has a small singular value as a rank degeneracy in R. The determination of whether R is rank deficient can be carried out by the condition estimator in [19].

If the QR factorization is employed as the initial orthogonal decomposition, the algorithm for the URV decomposition requires $O(2mn^2 + 7n^3/2 - 25nr^2/6)$ flops for computing the numerical rank of A from the triangular factor R (note that the previous algorithm can be stopped as soon as R is determined to be full rank, and the cost of estimating each right singular vector is $O(i^2)$ flops).

The initial triangularization in the first stage of the URV decomposition can be carried out by means of the efficient BLAS-3 QR factorization (LAPACK `xGEQRF`).

BLAS-3 computation kernels can be introduced in the second stage if we delay the application of Givens rotations, construct an aggregated transformation, and perform the updating by means of BLAS-3 kernels. Although the updating is more efficient, this algorithm presents a quite higher cost, due to the construction of the aggregated transformations.

Therefore, a different and also efficient approach has been selected, which presents a higher locality of data access. In our algorithm, the matrix is partitioned into $nb \times nb$ blocks, and the application of Givens rotations is performed by means of the usual BLAS-1 routine (xROT). Then, the application of the rotations is delayed so that each block is accessed only once.

4. Rank-revealing QR factorizations

The decomposition

$$AP = Q\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{3}$$

where P is an $n \times n$ permutation matrix, $Q \in \mathbb{R}^{m\times n}$ has orthonormal columns, and $R \in \mathbb{R}^{n\times n}$ is upper triangular, is said to be an RRQR factorization with numerical rank r iff $\kappa(R_{11}) \approx \sigma_1/\sigma_r$, $R_{11} \in \mathbb{R}^{r\times r}$, and $\|R_{22}\|_2 \approx \sigma_{r+1}$.

Whenever there is a well-defined gap in the singular value spectrum, the RRQR factorization reveals the numerical rank of A by having a well-conditioned leading submatrix R_{11}, and a trailing submatrix R_{22} of small norm.

4.1. The QR factorization with column pivoting

In [11] a column pivoting technique is introduced in the QR factorization [15] which in practice reveals the numerical rank of a matrix. The algorithm for the QRP can be stated as follows.

Algorithm QRP
for $j = 1{:}n$
  Determine k such that $\|A(j{:}m,k)\|_2 = \max_{i=j:n}\|A(j{:}m,i)\|_2$
  Permute the jth and the kth columns of A
  Compute a Householder reflector $Q_j \in \mathbb{R}^{m-j+1\times m-j+1}$, such that
    $Q_jA(j+1{:}m,j) = (\|A(j+1{:}m,j)\|_2, 0, \ldots, 0)^{\mathrm{T}}$
  Apply the Householder reflector to the matrix:
    $R(1{:}j,j) \leftarrow Q_jA(1{:}j,j)$
    $A(j{:}m,j+1{:}n) \leftarrow Q_jA(j{:}m,j+1{:}n)$
end for

In this algorithm, the computed triangular factor $\tilde{R}$ is the exact factor R of a matrix $A + \Delta A$, where $\|\Delta A\|_2 \leqslant c\epsilon\|A\|_2$, and c is a constant that depends on m and n [15].

This algorithm requires $O(4mnr - 2r^2(m+n) + 4r^3/3)$ flops if Q is not constructed.

The bulk of the computational work in the QRP is performed in the matrix update stage, which relies on matrix–vector operations and rank-1 updates

(BLAS-2). However, to arrive at a block QR factorization algorithm, we would like to avoid updating part of A until several Householder reflectors have been computed, and then apply these transformations by means of BLAS-3 operations [25]. This strategy is difficult to implement in the QRP, since a vector with the norms of the columns must be updated before the next pivot column can be chosen.

Recently, a new procedure has been proposed for computing the QR factorization with column pivoting, that allows the use of block reflectors, while maintaining the pivoting sequence and the numerical properties of this approach [23]. The idea behind the algorithm is the following: For each consecutive *nb* steps, in each step only one row and one column of the matrix is updated, leaving the rest to be updated at the end of the *nb* steps with a block update, namely, a rank-*nb* update. The new subroutine has been named `xGEQP3` and it will appear in the next public release of LAPACK.

4.2. A block QR factorization with restricted pivoting

In this subsection we describe a variant of the QR factorization which is more efficient on current architectures and besides produces an approximate rank-revealing triangular factor.

The "restricted" block pivoting algorithm proceeds in four phases:

Algorithm RRQR-Stage 1
Phase 1: Pivot column with largest two-norm into first position.
Phase 2: Compute a block QR factorization with restricted pivoting.
Phase 3: Carry out traditional pivoting among the columns rejected in phase 2.
Phase 4: Compute a QR factorization of the columns rejected in phase 3.

In the second phase of this algorithm, the scope of pivoting is limited, and thus it is possible to introduce block algorithms [2]. The idea is graphically depicted in Fig. 1. Consider that the columns to the left of the pivot window are already processed. Next, the following pivot column is selected from those inside the pivot window, this column is factorized, and the corresponding Householder reflector is only applied to the rest of the columns in the pivot window. Once several Householder reflectors are available, these are combined into a compact reflector YTY^{T}, which is applied then to the columns on the right. A more detailed description of this procedure can be consulted in [4].

Since only orthogonal transformations are involved in the algorithm, the computed triangular factor presents the same backward error as in the QRP. Moreover, it requires approximately the same number of flops as the QRP.

The factorization computed with this algorithm provides a good approximation to an RRQR factorization. However, due to the use of the restricted piv-

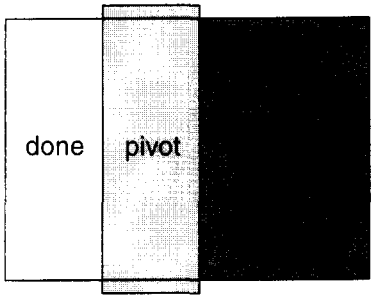

Fig. 1. Restricted pivoting of the QR factorization with local pivoting. All the pivoting is usually restricted to the window labeled "pivot".

oting, this QR factorization does not guarantee to reveal the numerical rank correctly. Therefore, a postprocessing stage is required which reveals the numerical rank from the triangular factor R.

4.3. Rank-revealing QR factorizations for triangular matrices

In [7,22] two theoretical approximations for computing the numerical rank of a triangular matrix are introduced. This triangular matrix can be obtained by means of an initial orthogonal decomposition like the QR factorization or the QRP. While the former is faster, it also produces a triangular factor which requires a more expensive postprocessing.

We have developed a variant of "Algorithm 3" in [22]. Given k and f, $0 < f \leqslant 1/\sqrt{k+1}$, the algorithm is guaranteed to halt and produce a factorization Eq. (3) that satisfies

$$\sigma_{\min}(R_{11}) \geqslant \frac{f}{\sqrt{k(n-k+1)}}\sigma_k(A), \tag{4}$$

$$\sigma_{\max}(R_{22}) \leqslant \frac{\sqrt{(k+1)(n-k)}}{f}\sigma_{k+1}(A). \tag{5}$$

The f-factor in this algorithm is used to prevent cycling of the algorithm. Thus, the higher f is, the tighter are the bounds in Eqs. (4) and (5), and the better the approximations to the kth and $(k+1)$st singular values of R. However, if f is too large, the algorithm requires more iterations, and because of round-off errors, it might present convergence problems (we used $f = 0.9/\sqrt{k+1}$ in our experimental studies).

Our implementation incorporates several improvements over this algorithm [4,5]. Specifically, an incremental condition estimator is employed to reduce the implementation cost, useless iterations are eliminated, and a more efficient application of Givens rotations is implemented.

In [7] an algorithm "Hybrid-III" is presented which computes an RRQR factorization with the following bounds:

$$\sigma_{\min}(R_{11}) \geq \frac{1}{\sqrt{k(n-k+1)}}\sigma_k(A), \tag{6}$$

$$\sigma_{\max}(R_{22}) \leq \sqrt{(k+1)(n-k)}\sigma_{k+1}(A). \tag{7}$$

In our experiments, we used a variant of this algorithm, described in [4]. Compared with the "Hybrid-III" algorithm, the variant presents several improvements; among these, a generalization of the f-factor technique is introduced to guarantee termination in the presence of rounding errors. The analysis in [24] proves that the new variant achieves the following bounds:

$$\sigma_{\min}(R_{11}) \geq \frac{f^2}{\sqrt{k(n-k+1)}}\sigma_k(A), \tag{8}$$

$$\sigma_{\max}(R_{22}) \leq \frac{\sqrt{(k+1)(n-k)}}{f^2}\sigma_{k+1}(A)\,, \tag{9}$$

where $0 < f \leq 1$. These bounds are very similar to Eqs. (6) and (7), except that the factor f^2 enters the equations (we used $f = 0.5$ in our experiments). Similar improvements to those of our implementation of "Algorithm 3" have also been carried out in our implementation of "Hybrid-III" algorithm.

5. Experimental results

In this section we report experimental results with the double-precision block-partitioned and parallel implementations of the algorithms presented in the previous sections. We consider the following codes:

`DGEQRF`: The block-oriented, BLAS-3 based implementation for computing the QR factorization without pivoting provided in LAPACK. This is not a rank-revealing subroutine since it does not produce any reordering of the columns, but it is included because it represents the optimal performance that might be achieved by the rank-revealing tools.

`DGEQPF`: The implementation for computing the QR factorization with column pivoting provided in LAPACK. This is not a block-oriented algorithm and, hence, its performance is independent of the block-size.

`DGEQP3`: A block-oriented and BLAS-3 based version of the subroutine DGEQPF, introduced in Section 4.1.

`DGEQPX`: An algorithm for computing an RRQR factorization which consists of two stages: the first one computes an approximate RRQR factorization (a QR with local pivoting). The second one is based on our variant of "Hybrid-III" algorithm.

DGEQPY: This algorithm is similar to the previous one, but in this case, the second stage is based on our variant of "Algorithm 3".

DGEURV: Our new block-oriented implementation of the algorithm for the URV decomposition.

DGESVD: The implementation of the SVD in LAPACK.

We carried out our experiments of our block-partitioned algorithms on a SUN Hypersparc-150 MHz and an SGI R10000-200 MHz.

We have also considered the following single-precision parallel codes:

SPGEQR2: BLAS-2 QR factorization.

SPGEQRF: BLAS-3 QR factorization.

SPGEQPF: BLAS-2 QRP factorization.

SPGEQPB: BLAS-3 QR factorization with local pivoting.

SPGEQPX: Parallel implementation of the QR factorization with column pivoting and Chandrasekaran and Ipsen's refinement algorithm.

Our parallel algorithms were tested on a cluster of 4 SUN Ultrasparc-170 MHz workstations. Single precision arithmetic was employed on this platform. In the implementation of our algorithms, we made extensive use of the available LAPACK infrastructure and we employed the vendor-supplied LAPACK and BLAS libraries.

5.1. Numerical reliability

We employed 18 different matrix types to evaluate the algorithms, with various singular value distributions and numerical rank ranging from 3 to full rank. Details of the test matrix generation are beyond the scope of this paper, yet they can be consulted in [4]. These types of matrices were specially designed to test several features of rank-revealing procedures: intensive column pivoting, behavior of the condition estimation in the presence of clusters of smallest singular values, etc.

We report in this section on results for square matrices of size $n = 1000$ on the SGI R1000-200 MHz, noting that identical qualitative behavior was observed for the same and smaller matrix sizes on the SUN Hypersparc and the SGI workstations. We decided to report on the largest matrix sizes because the possibility for failure in general increases with the number of numerical steps involved.

Fig. 2 reports the numerical results obtained with the 18 test matrices. We only report in this figure the results of the algorithms with a block size equal to 1 (left plot) and 20 (right plot). Closely similar results were obtained for other block sizes. This figure displays the ratio

$$\Theta := \frac{(\sigma_1/\sigma_r)}{\kappa(R_r)}, \tag{10}$$

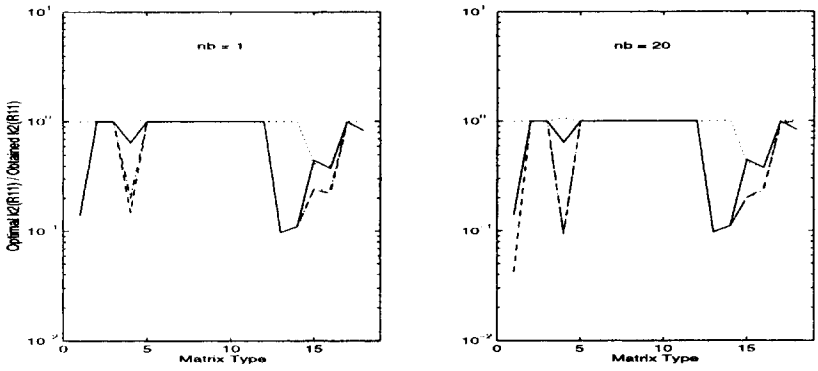

Fig. 2. Ratio between optimal and obtained condition number. Each plot contains the results for the 18 matrix types. Left plot for block size 1, right block for block size 20. The legend is the following: DGEQP3 (—), DGEURV (···), DGEQPX (- ·), and DGEQPY (- -).

where $\kappa(R)$ is the obtained condition number of R after the corresponding subroutine. Thus, Θ is the ratio between the ideal condition number and the obtained condition number of the leading triangular factor identified by the factorization. If this ratio is close to 1, the factorizations perform a good job capturing the "large" singular values of A. Obviously, subroutine DGESVD is not included since it provides the optimal behavior.

Fig. 2 shows that the block size usually does not play much of a role numerically, though close inspection reveals subtle variations.

The ratios for 1000×1000 are usually worse than those for smaller matrices. For example, the ratios for 100×100 matrices are smaller than 10^{-1}. In this case, the ratios are always smaller than 10^{-2} (two orders), which is usually enough for matrices with well-defined ranks because in these cases, the distance between the "large" singular values and the "small" singular values (those considered as null) is larger than two orders of magnitude.

Some of the "spikes" for test matrices, e.g., matrix types 1, 13, and 14, are not due to errors in our estimators. Rather, they show that all algorithms have difficulties when confronted with dense clusters of singular values. We also note that for some of the spikes the notion of rank is numerically ill-defined, since there is no sensible place to draw the line.

In summary, these results show that DGEQP3, DGEURV, DGEQPX, and DGEQPY are reliable algorithms for revealing the numerical rank. They produce rank-revealing factorizations whose leading triangular factors accurately capture the desired part of the spectrum of A, and thus provide reliable and numerically sensible rank estimates. Thus, the two-stage RRQR factorizations take advantage of the efficiency and simplicity of the QR factorization, yet they produce information that is almost as reliable as that computed by means of the more expensive SVD. Except for small variations, DGEQPX and DGEQPY

deliver identical results. Finally, the block-oriented code for computing URV factorizations obtains similar results.

5.2. *Performance of block-partitioned algorithms*

In this subsection we report on the performance of the LAPACK codes DGEQPF, DGEQRF, and DGESVD, as well as the new DGEQP3, DGEQPX, DGEQ-

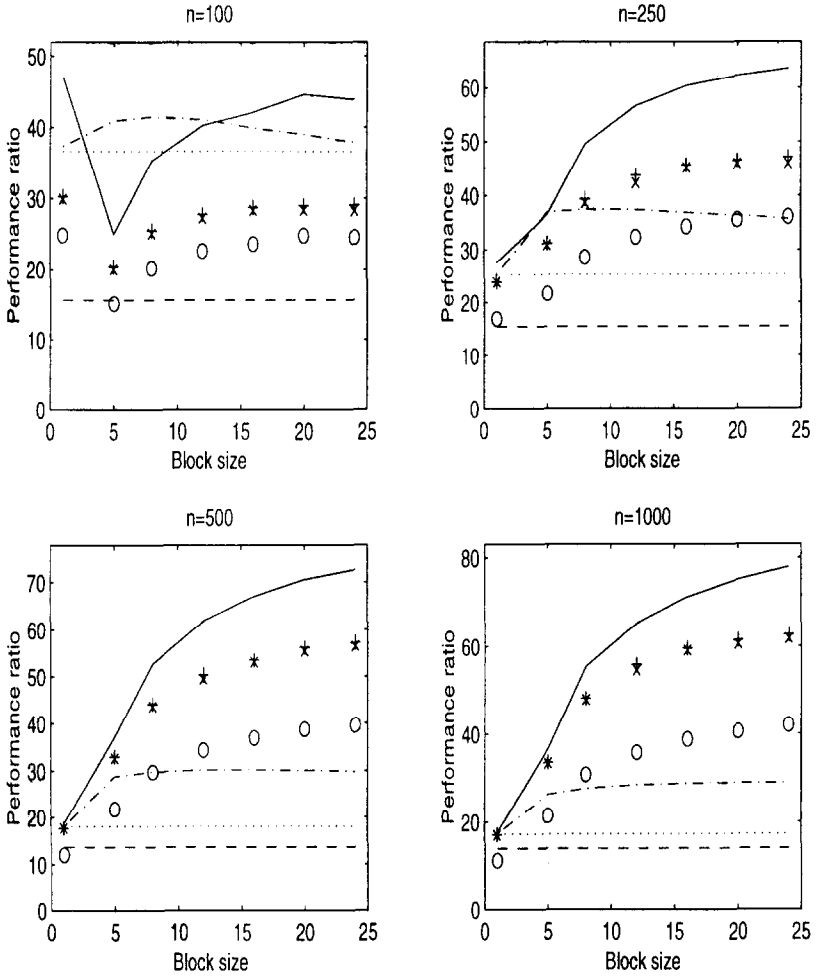

Fig. 3. Performance versus Block size on a SUN Hypersparc-150 MHz. Top left on 100 × 100 matrices, top right on 250 × 250 matrices, bottom left on 500 × 500 matrices, and bottom right on 1000 × 1000 matrices. The legend is as follows: DGEQRF (–), DGEQPF (···), DGEQP3 (- ·), DGEQPX (×), DGEQPY (+), DGESVD (- -), and DGEURV (○).

PY, and DGEURV codes. For these codes, as well as all others presented in this section, the Mflop rate was obtained by dividing the number of operations required for the unblocked version of DGEQRF by the runtime. This normalized Mflop rate readily allows for timing comparisons. We report on square matrix sizes 100, 250, 500, and 1000, using block sizes (nb) of 1, 5, 8, 12, 16, 20, and 24.

Figs. 3 and 4 show the Mflop performance (averaged over the 18 matrix types) versus block size.

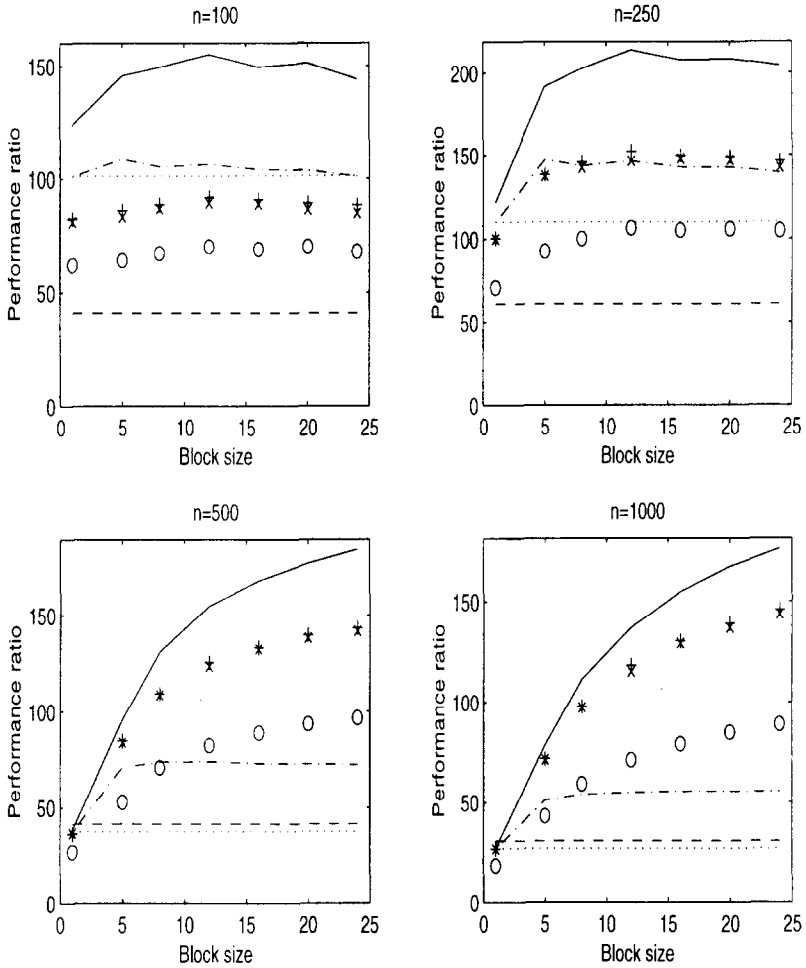

Fig. 4. Performance versus Block size on a SGI R10000-200 MHz. Top left on 100×100 matrices, top right on 250×250 matrices, bottom left on 500×500 matrices, and bottom right on 1000×1000 matrices. The legend is again DGEQRF (–), DGEQPF (···), DGEQP3 (- ·), DGEQPX (×), DGEQPY (+), DGESVD (- -), and DGEURV (○).

The figures show that the performances of the new algorithms for computing RRQR and URV factorizations are robust with respect to variations in the block size. The new tools for computing rank-revealing factorization are, except for small matrices, in average faster than usual LAPACK DGEQPF and, of course, LAPACK DGESVD. These results also show that DGEQPX and DGEQPY exhibit comparable performance, about three times as fast as DGEQPF. In contrast, subroutine DGEURV is about twice as fast as DGEQPF.

Small block sizes present two drawbacks. First, the performance is usually lower than the optimal one since the pipelined units and the cache memory are not fully used. Second, in the subroutines based on the QR with local pivoting, small block sizes can make the pivot window too small and, therefore, the rank may be underestimated in the preprocessing (QR factorization with local pivoting), which implies an extra effort of the postprocessing (RRQR factorization of triangular factor) to fix it.

On the other hand, large block sizes can also produce suboptimal performance, mainly on small matrices, since the ratio of columns factorized with BLAS-2 (those in the pivot window) and the number of columns factorized with BLAS-3 is too low.

Figs. 5 and 6 offer a closer look at the performance of the various test matrices. We chose $n = 250$ and $n = 500$ with $nb = 16$ as representative examples.

As shown in this figure, the performance of DGEQRF and DGEQPF does not depend strongly on the matrix type. There are some variations, but they be-

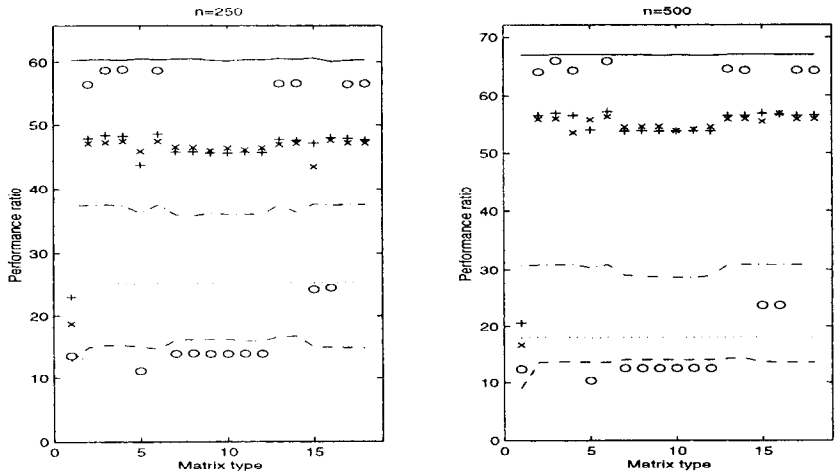

Fig. 5. Performance versus Matrix type on a SUN Hypersparc-150 MHz. Left on 250 × 250 matrices, right on 500 × 500 matrices. The legend is again: DGEQRF (–), DGEQPF (···), DGEQP3 (- ·), DGEQPX (×), DGEQPY (+), DGESVD (- -), and DGEURV (○).

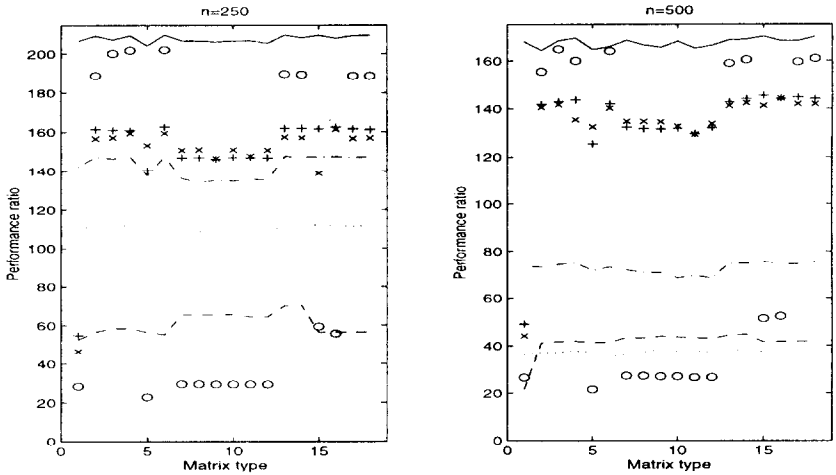

Fig. 6. Performance versus Matrix type on a SGI R10000-200 MHz. Left on 250 × 250 matrices, right on 500 × 500 matrices. The legend is again: DGEQRF (-), DGEQPF (···), DGEQP3 (- ·), DGEQPX (×), DGEQPY (+), DGESVD (- -), and DGEURV (○).

come minor as the matrix size grows. On the other hand, the DGESVD and the new subroutines present a higher variation rate.

Subroutine DGEURV obtains very good performances for matrices with rank equal to n or $n-1$. When the rank is smaller than those, the performances decrease very fast, e.g., for matrices with rank $3n/4$ (types 15 and 16) the performances are the same as LAPACK QR factorization with column pivoting, and for matrices with rank $n/2$ (types 1, 5, and 7–12) the performances are even worse than SVD.

We also see that the two-stage RRQR tools obtain the highest performances. For instance, these routines are around one to two times as fast as LAPACK DGEQPF and three times as fast as LAPACK DGESVD. The new code for computing the URV factorization is not so fast in most cases, and should be used only when it is known that the matrix rank is almost full.

5.3. *Performance of parallel distributed algorithms*

We have used a 2D block wrap data layout to develop our parallel algorithms. In this layout, the matrices are partitioned into blocks of dimension $r \times s$, and the blocks are distributed and stored locally in a $p \times q$ processor mesh (see Fig. 7).

This data layout defines the communication patterns and, therefore, the parallel algorithms. These algorithms are a direct parallel implementation of the numerical methods on the 2D layout. A 1×4 topology was employed in all

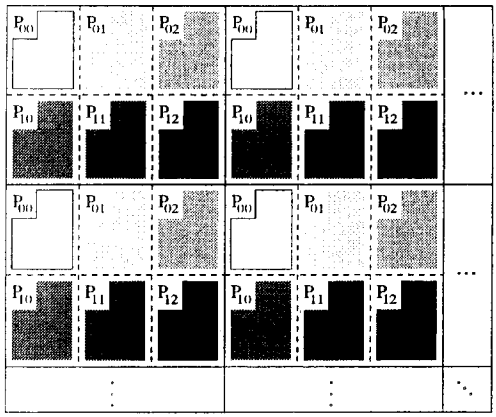

Fig. 7. Data layout in a mesh of 2×3 processors.

the cases. Thus, the block size of the distribution is $m \times s$, with $s = 1, 8, 10, \ldots, 50$, i.e., a cyclic column block data layout. Other topologies (2×2 and 4×1) were also tested but the performances of the algorithms were slightly lower since the number of involved processors was small.

In this section, the Mflop rate was again obtained by dividing the number of operations required for the unblocked version of DGEQRF by the runtime since this normalized Mflop rate readily allows for timing comparisons.

The first experiment is designed to analyze the effect of the block size on the performance of the parallel algorithms and, also, to compare the behavior of the different algorithms. Fig. 8 shows the average performance of the parallel algorithms on 400×400 matrices with 18 different matrix types and singular value distributions.

The results in the figure show that the highest performances are obtained, as we expected, by the QR factorization procedures, SPGEQRF and SPGEQR2. We are only interested in the performance of these routines because they present an optimal behavior. However, it is important to remember that these routines are not appropriate for rank-revealing purposes.

On the other hand, the lowest performance is achieved by the QRP procedure SPGEQPF. This is a BLAS-2 routine and, therefore, it has two disadvantages: it does not use the processors at full speed and the communications are fine-grained. Finally, the figures show that the two-stage parallel routine SPGEQPX achieves better performance than the QRP procedure. The difference between SPGEQPX and the parallel QR factorization with local column pivoting SPGEQPB is due to the overhead of the postprocessing step.

The speedups of the parallel algorithms were computed by comparing the total execution time of the sequential (single-precision) routines with the corresponding parallel (single-precision) implementations on the cluster of 4 SUN

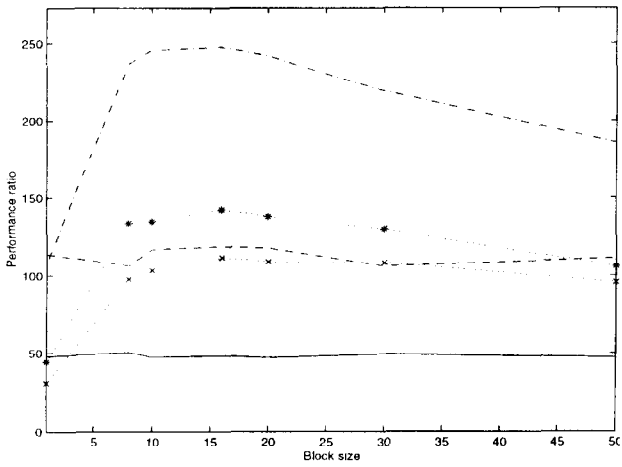

Fig. 8. Average performance versus Block size on the SUN Ultrasparc cluster. Solid line is employed for SPGEQPF; dotted line with symbol "*" for SPGEQPB; dotted line with symbol "×" for SPGEQPX; dashed–dotted line for SPGEQRF; and dashed line for SPGEQR2.

Ultrasparc processors for 4000×4000 matrices. The speedup of algorithm SPGEQPF is about 3.4. Nevertheless, algorithm SPGEQPF would obtain only a speedup about 1.4 if compared to our faster rank-revealing sequential solver SGEQPX. The speedup of algorithm SPGEQPX is 2.4.

Our following experiment is designed to study the performance of the algorithm when the size of the matrices is increased. Fig. 9 reports the performance

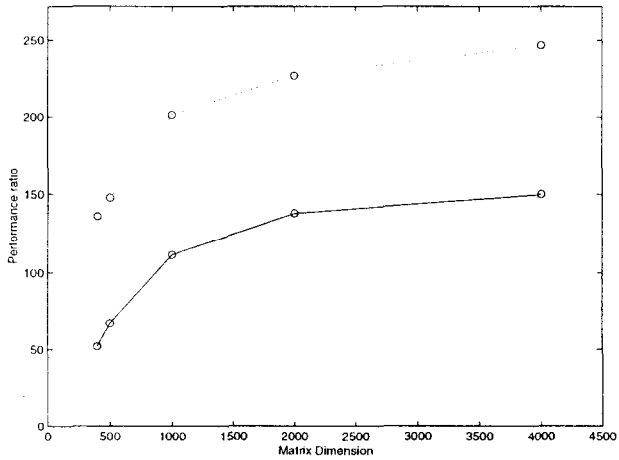

Fig. 9. Performance versus Matrix size on the SUN Ultrasparc cluster. Solid line with symbol "o" is employed for SPGEQPF and dotted line with symbol "o" for SPGEQPX.

of the parallel rank-revealing QR factorization algorithms (SPGEQPF and SPGEQPX) on full-rank random square matrices of sizes 400–4000. This figure shows a large difference in performance between the two-stage rank-revealing QR factorization SPGEQPX and the QR factorization with column pivoting. The gap between these algorithms is slightly increased for larger n.

6. Concluding remarks

We have presented an experimental analysis of the most well-known orthogonal factorizations for computing the numerical rank of dense matrices. The comparison includes the LAPACK SVD and QR factorization with column pivoting, a new block-partitioned implementation of the URV decomposition, and three efficient rank-revealing QR factorizations, which have been recently developed.

The new RRQR factorizations consistently outperform the traditional QR factorization with column pivoting on the SGI R10000 and the SUN Hypersparc platforms. Moreover, routines DGEQPX and DGEQPY provide reliable rank-revealing information. Routine DGEURV is highly efficient for matrices with rank close to full, while if the rank is low, the performance of this routine decreases very fast.

We have also presented parallel implementations of the rank-revealing QR factorizations on a distributed-memory architecture. The results showed that a numerical behavior closely similar to that of the traditional QR factorization with column pivoting (SPGEQPF) and a remarkable increase in the performance of routine SPGEQPX.

Acknowledgements

We express our gratitude to Christian H. Bischof for his interesting suggestions, ideas, and discussions. We also thank the referee for his/her valuable comments.

References

[1] E. Anderson, Z. Bai, C.H. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK User's Guide Release 2.0, SIAM, Philadelphia, 1994.

[2] C.H. Bischof, A block QR factorization algorithm using restricted pivoting, in: Proceedings SUPERCOMPUTING '89, Baltimore, ACM, New York, 1989, pp. 248–256.

[3] C.H. Bischof, M. Shroff, On updating signal subspaces, IEEE Trans. on Signal Processing 40 (1992) 96–105.

[4] C.H. Bischof, G. Quintana-Ortí, Computing rank-revealing QR factorizations of dense matrices, Argonne Preprint MCS-P559-0196, Mathematics and Computer Science Division, Argonne National Laboratory, 1996. To appear in ACM Trans math. Softw. 1998.
[5] C.H. Bischof, Codes for rank-revealing QR factorizations of dense matrices, Argonne Preprint MCS-P560-0196, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.
[6] T.F. Chan, An improved algorithm for computing the SVD, ACM Trans. on Math. Soft. 8 (1982) 72–83.
[7] S. Chandrasekaran, I. Ipsen, On rank-revealing QR factorizations, SIAM J. on Matrix Analysis and Applications 15 (1994) 592–622.
[8] J. Demmel, W. Kahan, Computing small singular values of bidiagonal matrices with guaranteed high relative accuracy, SIAM J. on Scientific and Statistical Computing 11 (1990) 873–912.
[9] J.J. Dongarra, J.R. Bunch, C.B. Moler, G.W. Stewart, LINPACK Users' Guide, SIAM, Philadelphia, 1979.
[10] L. Eldén, R. Schreiber, An application of systolic arrays to linear discrete ill-posed problems, SIAM J. on Scientific and Statistical Computing 7 (1986) 892–903.
[11] G.H. Golub, Numerical methods for solving linear least squares problems, Numerische Mathematik 7 (1965) 206–216.
[12] G.H. Golub, W. Kahan, Calculating the singular values and pseudo-inverse of a matrix, SIAM J. on Numerical Analysis 2 (1965) 205–224.
[13] G.H. Golub, V. Klema, G.W. Stewart, Rank degeneracy and least squares problems, Technical Report TR–456, University of Maryland, Dept. of Computer Science, 1976.
[14] G.H. Golub, P. Manneback, P.L. Toint, A comparison between some direct and iterative methods for certain large scale geodetic least-squares problem, SIAM J. on Scientific and Statistical Computing 7 (1986) 799–816.
[15] G.H. Golub, C.F. Van Loan, Matrix Computations, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.
[16] T.A. Grandine, An iterative method for computing multivariate C^1 piecewise polynomial interpolants, Computer Aided Geometric Design 4 (1987) 307–319.
[17] T.A. Grandine, Rank deficient interpolation and optimal design: An example, Technical Report SCA–TR–113, Boeing Computer Services, Engineering and Scientific Services Division, 1989.
[18] P.C. Hansen, S. Takishi, S. Hiromoto, The modified truncated SVD-method for regularization in general form, SIAM J. on Scientific and Statistical Computing 13 (1991) 1142–1150.
[19] N.J. Higham, A survey of condition number estimation of triangular matrices, SIAM Review 29 (1987) 575–596.
[20] S.F. Hsieh, J.R. Liu, K. Yao, Comparisons of Truncated QR and SVD methods for AR spectral estimations, in: Proceedings SVD and Signal Processing II, Elsevier, Amsterdam, 1991, pp. 403–418.
[21] J. Moré, The Levenberg-Marquardt algorithm: Implementation and theory, in: G.A. Watson (Ed.), Proceedings of the Dundee Conference on Numerical Analysis, Springer, Berlin, 1978.
[22] C.-T. Pan, P.T.P. Tang, Bounds on singular values revealed by QR factorization, Argonne Preprint MCS-P332-1092, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.
[23] G. Quintana-Ortí, X. Sun, C.H. Bischof, A BLAS-3 version of the QR factorization with column pivoting, Argonne Preprint MCS-P551-1295, Mathematics and Computer Science Division, Argonne National Laboratory, 1995. To appear in SIAM J. Sci. Comp., 1998.
[24] G. Quintana-Ortí, E.S. Quintana-Ortí, Guaranteeing termination of Chandrasekaran and Ipsen's algorithm for computing rank-revealing QR factorizations, Argonne Preprint MCS-

P564-0196, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[25] R. Schreiber, C.F. van Loan, A storage efficient WY representation for products of householder transformations, Scientific and Statistical Computing 10 (1989) 53–57.

[26] G.W. Stewart, An updating algorithm for subspace tracking, Technical Report CS-TR-2494, University of Maryland, Department of Computer Science, 1990.